# Architektur und Arbeitsweise eines Web-Browsers

Hauptseminar
„Datenbank- und Informationssysteme“
WS 2000

Markus Legner

Betreuer: Dipl.-Inf. Aiko Frank

14.01.2001

# Inhaltsverzeichnis

# 1 Einleitung

Was verbindet einen Browser mit Datenbank- oder Informationssystemen?
Selbst der Laie erkennt recht schnell, daß ein Browser, für sich alleine, recht wenig mit der Verwaltung von Daten und Informationen zu schaffen hat. Worin besteht also der Zusammenhang, der den Bereich *Web-Browser* für das Oberthema dieses Seminars relevant macht?

Ein IS[1] hat nur einen Nutzen, wenn die verwalteten Daten in geeigneter Weise genutzt werden können. Das heißt, ein Benutzer muß

- die relevanten Daten,
- am richtigen Ort,
- zur richtigen Zeit,
- auf einfache Art und Weise

abrufen können.

Hierfür bieten bereits existierende Webtechnologien sehr gute Voraussetzungen. Ein Web-Browser bietet eine gewohnte, im Grunde relativ einfach zu bedienende Oberfläche. Außerdem kann durch den flexiblen, nicht fest verdrahteten Zugriff über das Netzwerk von verschiedenen Standorten, zu beliebigen Zeiten zugegriffen werden. Ein weiterer positiver Effekt, bei der Nutzung von Internettechnologie, ist die Verminderung von Kosten für Kauf und Installation von speziellen Anwendungen. Dies ist die Folge daraus, daß ein Browser für den Datenzugriff ausreichend ist.

Diese Vorteile machen sich viele Unternehmen verstärkt zunutze. Heutzutage findet sich kaum noch ein größeres Unternehmen, das nicht ein eigenes *Intranet* betreibt, um die Informationsinfrastruktur und somit die Produktivität und Effizienz der Mitarbeiter zu verbessern. Nahezu alle wichtigen Produzenten von Unternehmenssoftware haben den Trend ebenfalls erkannt, und bieten mittlerweile Web-Schnittstellen für ihre Produkte.

Entscheidend ist, daß ein Browser nicht nur die Standardfunktionen übernehmen kann. Dies sind Funktionen wie das Darstellen von statischen HTML- und XML- Seiten, die Unterstützung von Links oder die Fähigkeit Bilder darstellen zu können. Ein Browser muß auch Erweiterungsmöglichkeiten bieten, so daß z.B. komplexere Bedienschnittstellen, oder die Verarbeitung proprietärer Datenformate realisiert werden können. Dies wird u. a. ein Thema dieser Arbeit sein.

# 2 Browser allgemein

## 2.1 Logische Architektur eines Browsers

Es lassen sich - auf einer sehr abstrakten Ebene - die folgenden logischen Komponenten eines Browsers unterscheiden:

Der *Controller* ist die zentrale Komponente, die die Abläufe im Browser regelt und überwacht. In ihm werden auch alle Eingaben des Benutzers verarbeitet. Beim Browser ist dies normalerweise auf Maus- und Keyboardevents beschränkt. Ein Beispiel für eine Benutzeraktion wäre das Anfordern einer neuen URL[2] durch den Benutzer.

[1] Informationssystem
[2] Uniform Ressource Locator

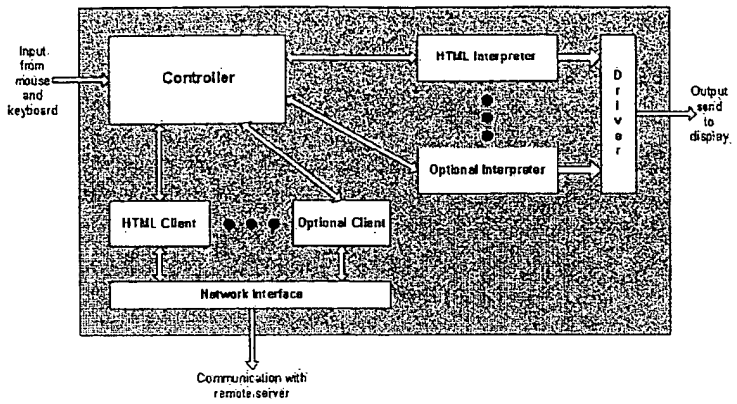



Abbildung 1: Der logische Aufbau eines Webbrowsers

Der *HTML Client* ist dafür verantwortlich, einen Teil der dem Controller mittgeteilten Aufträge auszuführen. Im Falle des *HTML CLient* betrifft das vor allem das Anfordern von HTML Seiten über das Netzwerk. Der *HTML Client* bedient sich dazu der Netzwerkkomponente. Sie kontaktiert den entsprechenden *Webserver* über HTTP [3] und liefert die Ergebnisse zurück.

Der *HTML Interpreter* tritt dann in Aktion, wenn die Informationen aus dem neu empfangenen HTML-Dokument grafisch auf dem Monitor darzustellen sind. Der Interpreter wird vom Controller aufgerufen, wenn ein neues Dokument empfangen wurde. Er parst den HTML-Code und liefert die Ergebnisse an den Grafiktreiber Wie die Erzeugung der grafischen Ausgabe abläuft wird in einem der folgenden Kapitel genauer besprochen.

Neben dem HTML Client finden sich in heutigen Browsern noch eine Vielzahl anderer Clients. Beispiele hierfür wäre der *FTP*[4] *Client*, oder ein Gopher Client[5].

Wie es verschiedene Clients geben kann, so kann es auch verschiedene Interpreter - zum Darstellen von Dokumenten, die in verschiedenen Sprachen vorliegen - geben. Ein Beispiel hierfür wäre ein *XML*[6] *Interpreter*, der XML-Dokumente verarbeiten kann.

## 2.2 Browserkonzepte

### 2.2.1 DOM und DHTML Object Model

Ehemals waren alle HTML-Seiten statischer Natur. Das bedeutet, die Seite wurde über das Netz gelesen, ausgewertet und einfach auf dem Bildschirm ausgegeben. Am Anfang war die Repräsentation der Seiten im Browser total „flach". Das bedeutet es bestand keine Möglichkeit einzelne Teile der Seite zu ändern, z. B. waren die *Tags* nicht direkt adressierbar. Ein HTML-Dokument wurde genau einmal abgearbeitet und dargestellt.

Mit der Weiterentwicklung des Internets reichten diese Datenstrukturen aber nicht mehr aus, um die gewünschte Interaktivität zu erreichen. Die Konsequenz war die Standardisierung des Datenmodells durch das W3C[7]. Das Resultat hieraus ist das *Document Object Model* (DOM).

[3] Hypertext Transfer Protocol
[4] File Transfer Protocol
[5] Veraltetes Protokoll, das zum Suchen nach Themen im Internet verwendet wurde, und teilweise noch wird.
[6] Extended Markup Language
[7] World Wide Web Consortium[6]

**DOM** Das DOM ist eine plattform- und sprachunabhängige Schnittstelle. Sie erlaubt es über Skriptsprachen auf den Inhalt, die Struktur und den Stil eines Dokumentes zuzugreifen, und diese zu ändern. Das DOM umfasst ein Modell, das beschreibt, wie eine standardisierte Menge von Objekten, die HTML- und XML-Dokumente repräsentieren, kombiniert werden kann. Weiter beinhaltet es noch eine Schnittstelle, die es ermöglicht diese Objekte zu adressieren und sie zu manipulieren. Ein Hauptvorteil des DOM ist die Möglichkeit, auf alles im Dokument Zugriff zu haben, was beliebige Aktualisierungen erlaubt. Zusätzlich kann auf einzelne Fragmente des Dokuments gezielt zugegriffen werden.
Die DOM-Spezifikation ist in mehrere Level und darin noch in verschiedene Bereiche gegliedert. Im folgenden werden kurz zwei Bereiche aus dem ersten Level beschrieben:

- **Document Object Model Level 1 (Kern)**
  Diese Stufe definiert eine minimale Menge von Objekten und Schnittstellen um auf Dokumenten-Objekte zugreifen zu können und sie zu bearbeiten. Die Kern-Stufe bietet eine allgemeine API um bereits *geparste* HTML- und XML-Inhalte verarbeiten zu können. Außerdem erlaubt sie es Dokumente komplett im Hauptspeicher aufzubauen. Das persistente Speichern bleibt dem Programmierer überlassen.

- **Document Object Model Level 1 (HTML)**
  Die HTML-Stufe erweitert den Kern um Objekte und Methoden, die spezifisch in HTML-Dokumenten Verwendung finden. Der Kern enthält allgemein Spezifikationen, die benötigt werden um hierarchische Strukturen, Elemente und Attribute zu bearbeiten. Hier - im HTML-Bereich - wird Funktionalität definiert, die spezifischen HTML-Elementen zugeordnet ist. Sie ist also nicht global nutzbar.

Aktuell ist zur Zeit Level 3, wobei jedes Level ca. sechs Bereiche hat, so daß hier aus Platzgründen nicht darauf eingegangen werden kann. Nach meinem Wissens implementieren die aktuellen Browsergenerationen bestenfalls Teile von Level 2. Die Spezifikationen finden sich auf der Homepage des W3C[6].

**DHTML Object Model** Das *DHTML Object Model* ist eine schwächer ausgeprägte, weniger strukturierte Form des DOM. Während das DOM wie erwähnt vom W3C standardisiert ist, ist das jeweilige DHTMLOM abhängig von der Spezifikation des Browser-Herstellers. Es gestattet Autoren direkten, programmierfähigen Zugriff auf die einzelnen Komponenten ihres Web-Dokumentes - von einzelnen Elementen bis hin zu *Containern*. Der Zugriff über das DHTMLOM kombiniert mit dem *Event Model* erlaubt z. B. die folgenden Aktionen: Reaktion auf Benutzereingaben, Ausführung von Skripten „on-the-fly“ oder die Anzeige neuer Inhalte ohne den Server zu kontaktieren.

Beide Modelle stimmen darin überein, daß jedes Element und jedes Attribut über Skript erreichbar ist. Es gibt aber auch ein paar Unterschiede. Das DOM ist eine Weiterentwicklung des DHTMLOM mit dem Zweck mehr Robustheit zu erreichen. Im Gegensatz zum DHTMLOM ist das DOM viel stärker strukturiert, und es bietet eine logische Schnittstelle für den Zugriff und das Ändern der Elemente und Attribute.

Die Manipulation des Dokument-Baums mittels DOM hat folgende Vorteile:

- Ein Teil des Baumes kann an eine andere Stelle geschoben werden, ohne den Inhalt löschen und wieder hinzufügen zu müssen.

- Neue erstellte Dokumente können an jeder Stelle im Baum eingehängt werden.

- Einzelne Zweige können innerhalb eines Dokumenten-Fragmentes organisiert und verändert werden. Erst danach werden die Objekte wieder in den Baum eingefügt.

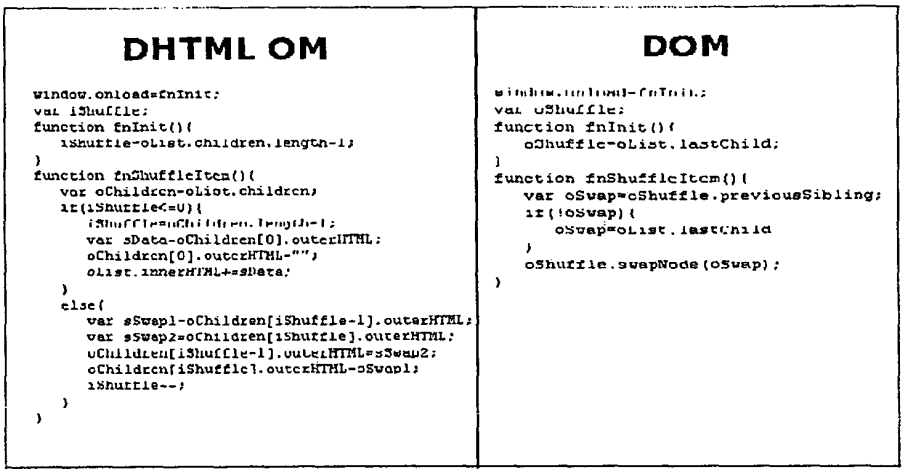



Abbildung 2: Vergleich zwischen DHTML- und DOM-Programmierung [2]

### 2.2.2 client-basierte Skriptausführung

Ein Skript ist ein Programm - in einer Skriptsprache geschrieben - , das nicht compiliert wird, sondern das zur Laufzeit von einem Interpreter ausgeführt wird. Wie oben bereits angesprochen geben Skripte einer Web-Seite Dynamik, da es über die beschriebenen Modelle möglich ist den Inhalt der Seite „zu programmieren".

Skripte in Web-Seiten werden auf dem Client/Web-Browser ausgeführt und ermöglichen somit mehr Funktionalität und sogar Interaktivität zu erreichen. Server-seitig erlauben Skripte hingegen dynamisch HTML-Seiten zu erzeugen (welche wieder client-seitige Skripte enthalten können).
Im Internet gibt es prinzipiell zwei mögliche Orte den Skriptcode in einer Seite auszuführen: Webserver oder Browser. Hier soll nur die 2. Variante betrachtet werden.

Die Skriptinterpreter im Browser werden normalerweise als *Script-Engines* (SE) bezeichnet. Beim Laden der Seite läßt der Browser den Skriptcode vom korrekten SE interpretieren. Wird in einer Seite ein *Event*[8] ausgelöst, so führt der Browser den dem *Event* zugeordneten Skriptcode aus.

Ein Problem der client-basierten Skriptausführung ist natürlich die Kompatibilität zwischen den Browsern, was deren Ausstattung betrifft. Wird eine Skriptpassage vom eigenen Browser korrekt interpretiert, so kann man deswegen noch lange nicht annehmen, daß dies tatsächlich jeder Browser kann. Dieses Problem tritt natürlich beim server-basierten Ansatz nicht auf, was auch dessen großer Vorteil ist.

### 2.2.3 Java Virtual Machine (VM)

Um aktive Java-Inhalte zu verwenden bedarf es eines java-fähigen Browsers[9]. Java Applets, die in einer Webseite enthalten sind, werden immer innerhalb einer speziellen Komponente, der sogenannten *Java Virtual Machine* ausgeführt. Die VM ist durch die Schnittstellen-Spezifikation von Sun festgelegt. Mittlerweile gibt es für fast alle Plattformen entsprechende VMs. Es ist jederzeit möglich eine eigene VM zu implementieren, solange sie die spezifizierte Schnittstelle umsetzt. So verwendet der Internet Explorer beispielsweise eine eigene JVM, die auf ActiveX (4.1) basiert.

[8]Engl.: Ereignis. Eine Nachricht, die einer bestimmten Komponente mitteilt, daß eine bestimmte Aktion ausgeführt wurde.

[9]z. B. die aktuellen Browser von Netscape oder Microsoft

### 2.2.4 MIME-Typen

*MIME* steht für *Multipurpose Internet Mail Extension*. MIME ist eine standardisierte Methode für die Organisation von verschiedenen Dateiformaten. Dabei werden Dateien nach ihrem jeweiligen MIME-Typ, der das Format beschreibt organisiert. Ließt der Browser eine Datei von einem Webserver, so liefert der Server den MIME-Typ der Datei mit. HTTP-Server verwenden z. B. die HTML-Formatierung („text/html"). Der Browser bestimmt anhand des MIME-Typs, ob er die Daten von sich aus verarbeiten kann, oder ob zusätzliche Software benötigt wird, die er in diesem Fall dann, auf dem System, suchen würde.

Zwar unterstützen viele Plattformen wie zum Beispiel *Windows* Standardverknüpfungen zwischen Dateinamenerweiterungen und Dateitypen. Doch ist diese Verknüpfung nicht eindeutig genug. So kann die Dateinamenerweiterung „.doc" beispielsweise ein Word-Dokument, ein FrameMaker-Dokument oder eine einfache Textdatei bedeuten. Das System der Mime-Typen hat die Aufgabe, solche Zweideutigkeiten zu beseitigen und den beteiligten Programmen (WWW-Server-Software, WWW-Browser) ein eindeutiges Identifizierungsschema für den Datentyp von Dateien bereitzustellen.

MIME-Typen werden nach folgendem Schema angegeben:
`Kategorie/Unterkategorie`

Kategorien sind z.B. *text*, *image*, *audio*. Unterkategorien von *text* sind beispielsweise *plain* (Datei ist eine reine Textdatei) oder *html* (Datei ist eine HTML-Datei). Unterkategorien von *image* ist beispielsweise *gif* (Datei ist eine Grafik im GIF-Format).

### 2.2.5 Cookies

Ein grundlegendes Problem des Internets ist die Zustandslosigkeit des Protokolls, auf dem eigentlich das ganze Netz basiert: HTTP. Das bedeutet, jeder neue Seitenaufruf ist für Webserver und -browser völlig isoliert. Wird eine Seite zeitlich getrennt zwei mal aufgerufen, so weiß der Webserver beim 2. Aufruf normalerweise nichts von dem ersten Besuch der Seite. Als das Internet, wie wir es heute kennen, konzipiert und in Betrieb genommen wurde, war dieses Manko noch nicht so gravierend wie heute mit neuen Technologien, wie sie z. B. für E-Commerce benötigt werden.

Die Lösung des Problems sind u. a. *Cookies*; dt.: Kekse. Das sind kleine Textdateien, die vom Browser in einem bestimmten, ausgezeichneten Verzeichnis abgelegt werden[10]. In den Cookies werden Informationen abgespeichert, die ein Webserver zu einem späterern Zeitpunkt eventuell benötigt. Diese Informationen sind nur vom „besitzenden" Webserver abrufbar, und der Benutzer kann die Verwendung von Cookies expizit verbieten.

Cookies müssen nicht zwingend auf der Festplatte des Browsers abgelegt werden. Das ASP[11] - *Sessionmanagement* verwendet z. B. Cookies, die im Speicher des Client-Rechners abgelegt werden. Bei nachfolgenden Anfragen an den selben Server werden dann die Cookie-Informationen wieder mitgeschickt.

### 2.2.6 Cache

Die Funktion des *Caches* unterscheidet sich im Browser nicht von anderen Systemen. Deshalb wird hier nur kurz darauf eingegangen.
In einem Netzwerk ist es klar, daß Zugriffe über das Netzwerk um ein Vielfaches teurer sind, als lokale Datenzugriffe. Der Cache im Browser hat die Funktion eine lokale Kopie jeder gelesenen Seite zu halten. Soll eine Seite neu angefordert werden, so prüft der Browser zunächst, ob die Seite nicht

[10] unter Win 2000 z. B. im „Cookies"-Verzeichnis, im Homeverzeichnis des Benutzers
[11] Active Server Page

lokal im Cache liegt. Ist dies der Fall, so wird nur die lokale Kopie verwendet, und der teure Zugriff über das Netz wird vermieden. Wird die Seite nicht im Cache gefunden, so ist der Zugriff über das Netzwerk natürlich unvermeidbar.

Die Caching-Funktion des Browsers läßt sich auf drei Weisen abschalten. Am einfachsten ist die Caching-Konfiguration über das Menü des Browsers. Die zweite Möglichkeit, ist das Vermeiden des Zwischenlagerns mittels eines speziellen HTTP *Cache-Control Headers*. Zu guter letzt kann eine *HTTP-EQUIV META Tag* in der HTML-Datei verwendet werden, dessen Interpretation allerdings browserabhängig ist.

## 2.3 Der Prozeß der Seitenerzeugung

### 2.3.1 Parser Komponenten

Die folgenden Komponenten sind auf dem Weg vom *HTTP-Stream* zum DOM beteiligt:

- **Scanner** Der Scanner liefert eine API[12], die z. B. folgende Aktionen gestattet: Zugriff auf Zeichen im *Input Stream*, Auffinden bestimmter Zeichensequenzen, Anpassen der eingehenden Daten und das Überspringen unerwünschter Daten.

- **Parser** Der Parser ist dem Scanner nachgelagert. Er kontrolliert und koordiniert die Aktivitäten der anderen Komponenten im Layoutsystem. Während die restlichen Komponenten in Abhängigkeit vom Typ der Quelldatei dynamisch ausgetauscht werden müssen, ist es selten notwendig den Parser zu ändern. Der Parser ist auch zuständig für die Bildung von *Tokens*[13]. Die grammatikalischen Regeln eines Dokumenttyps werden dynamisch angepasst.

- **DTD-Komponente**[14] Sie beschreibt die Regeln für korrekte Dokumente, abhängig vom Dokumenttyp. Für HTML legt sie die Menge von *Tags*, die zugeordneten Attribute und die Verschachtelungsregeln für *Tags* fest. Durch die Kapselung der Strukturinformationen wird das System dynamischer.

- **„Auffangbecken"** Diese letzte Komponente ist eine einfache API, die Teile des vom Parser erstellten Parse-Baumes als Eingabe akzeptiert, und daraus, entsprechend dem zugrundeliegende *Object Model*, das Objekt, das das Dokument repräsentiert, konstruiert. Der Parser hat von sich aus keine Informationen über das OM.

# 3 Mozilla / Netscape Navigator

Weitere Informationen zu diesem Kapitel finden sich im Internet. Die Homepage des *Mozilla*-Projekts [4] bietet vor allem Informationen zu konkreten Implementierungsfragen. Demgegenüber sind die Informationen auf der *Netscape-Developer*-Seite [5] eher übergeordnet. Hier gibt es Informationen zu verschiedenen SDK's[15] und APIs.

## 3.1 Plug-in API

Die normalen Standardsprachen im Web sind HTML oder XML, d.h. normalerweise lädt ein Browser eine HTML- oder XML-Seite und er weiß, wie er diese darzustellen hat. Natürlich ist die reale Welt nicht auf diese beiden Dateiformate beschränkt, es mußte also die Möglichkeit geschaffen werden viele verschiedene Dateiformate darzustellen. Man könnte zwar hergehen, und den Browsercode bei jedem

[12] Application Programming Interface
[13] Z. B. die Konvertierung einer eingehenden unstrukturierten Zeichenkette in HTML *Tags*.
[14] Document Type Definition
[15] Software Development Kit

neuen Format erweitern, in der Praxis wäre das aber zu ineffizient.
Der erste Lösungsansatz sind die sogenannten *Helper Applications*. Das heißt nichts anderes, als daß der Browser bei einem ihm fremden Dateiformat eine vollständig außerhalb des Browser-Prozesses ablaufende, externe Applikation aufruft, die das Format darstellen kann. Die Wahl der richtigen Applikation ist vom MIME-Typ der Datei abhängig. Die Lösung hat natürlich den Nachteil, daß der Browser die Kontrolle an die Applikation abgeben muß und somit keinen Einfluß mehr auf nachfolgende Aktionen hat.
Der zweite Lösungsansatz sind die *Plug-ins*.

Ein Plug-in ist ein separates Modul, das sich verhält, als wäre es ein Teil des Browsers. Die Plug-in API kann dazu verwendet werden, Plug-ins zu erstellen, die dem Browser interaktive und multimediale Fähigkeiten verleihen. Die erstellten Plug-ins können einem oder mehreren MIME-Typen zugeordnet werden. Der Browser erstellt die Zuordnungstabelle beim Programmstart. Dabei werden alle Plug-ins, die im ausgewiesenen „Plug-in-Ordner" liegen, nacheinander durchlaufen und abgefragt.

Wird eine Webseite aufgerufen, die eingebettete „Plug-in-Daten" enthält so checkt der Browser zunächst, ob der MIME-Typ einem Plug-in zugeordnet ist. Ist ein passendes Plug-in installiert, so passiert folgendes: der Code wird in den Hauptspeicher geladen, das Plug-in wird initialisiert und schließlich wird eine neue Instanz des Plug-ins erzeugt. Verläßt der Benutzer die Webseite, oder wird das Browserfenster geschlossen, so wird die Plug-in Instanz zerstört.

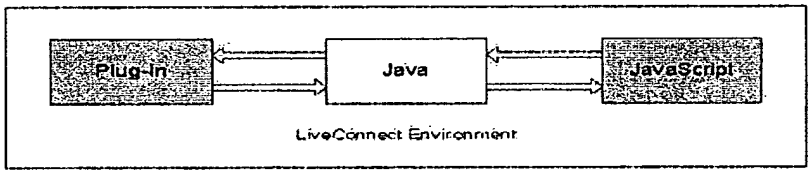
Die Plug-in API soll folgende Ziele erreichen:

- Erweiterung der Fähigkeiten des Browsers, indem *Inline Viewer*[16] erstellt werden können. Diese unterstützen Dateiformate, die der Browser nicht von sich aus unterstützt.
- Die Plug-in API, ist einfach und klein. Dies ermöglicht es relativ einfach existierende Bibliotheken wieder zu verwenden, oder bestehende Applikationen für den Einsatz im Web zu konvertieren[17].

Plug-ins können außerdem das *Java Runtime Interface* (JRI) verwenden, um auf die Java-Umgebung zugreifen zu können. Die Kommunikation mit und über Java, mittels *JavaScript* funktioniert über eine *LiveConnect*-Verbindung (3.2).

## 3.2 LiveConnect

Seit dem Navigator 3.0 bietet Netscape den Entwicklern die Möglichkeit Java und JavaScript für die Interaktion mit einer Webseite zu verwenden. Dies könnte z. B. über eine Plug-in Komponente oder über die Benutzerschnittstelle geschehen. So können einige Plug-ins über JavaScript gesteuert werden.
Obwohl Plug-ins mit Java wie auch Java mit JavaScript direkt interagieren können, interagieren JavaScript und Plug-ins nicht direkt miteinander. *LiveConnect* bietet die Schnittstelle für die Kommunikation zwischen allen drei Technologien.




[16]Kleine Programme, die die Daten im Browserfenster darstellen.
[17]z. B. das *Adobe Acrobat Plugin* zum darstellen von PDF-Dateien

Soll ein Plug-in *LiveConnect* unterstützen, so muß der Standard-C-Code etwas erweitert werden. Zusätzlich zum eigentlichen Code müssen die *Plug-in-API*-Methoden implementiert werden, die es dem Plug-in erlauben mit dem *Java Object Model* zu interagieren.

## 3.3 Gecko Layout Engine

Der *Gecko Layout Engine* (GLE) basiert auf einem plattformunabhängigen Objekt-Modell: *XP-COM*[18]. Wie der Name schon sagt handelt es sich dabei um eine plattformunabhängige Weiterentwicklung des Standard-*COM*. Alle Komponenten des GLE benutzen *XP-COM* für die Kommunikation untereinander.

Die Hauptziele bei der Entwicklung des GLE waren einen kleinen, erweiterbaren und schnellen Layout Engine zu schaffen. Der GLE kann um beliebige Dokumenten-Formate erweitert werden. Dazu muß ein entsprechender *Document Viewer*[19] geschaffen werden und dem *Parser* muß ein entsprechendes Objekt zur Verfügung gestellt werden, daß die grammatikalischen Regeln beinhalten..

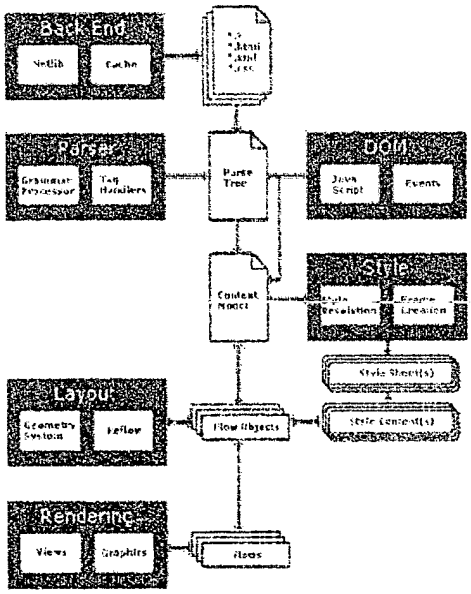



Abbildung 3: Gecko basiert auf sechs Subsystemen. Vom Netzwerk-*Back End* bis zum *Rendering Engine*, der die graphische Ausgabe generiert.[1]

Die Funktionsweise der Komponenten wurde teilweise schon bei den Parser-Komponenten (2.3.1) umrissen. Hier wird noch etwas detaillierter darauf eingegangen.

**Back End** Das ist die Netzwerkkomponente. Wird ein neues Dokument empfangen, so wird zunächst der korrekte *MIME-Typ* bestimmt. Dieser Prozeß wird von ein paar Seiten unterstützt: Neben der *URL*[20] und dem empfangenen *HTTP Header* werden in manchen Fällen auch direkt Bytes am Anfang des Dokuments ausgewertet. Der *MIME-Typ* wird verwendet um den passenden *Document Viewer*

[18] Extended Platform Component Object Model
[19] ist in der Lage den Dokumenttyp zu beschreiben und zu *rendern*
[20] Uniform Ressource Locator

(DV) dynamisch zu generieren. Standardmäßig beinhalted Gecko DVs für HTML, XML und Grafiken.
Die empfangenen Daten werden anschließend zum *Parser* weitergeschickt.

**Parser** Der *Parser* erstellt den *Parse-Tree* (PT). Der PT beschreibt den Inhalt des erhaltenen Fragments eines Dokuments symbolisch. Die Hauptaufgaben des PT sind die Interpretation der empfangenen Daten, die Prüfung der strukturellen Korrektheit und das Entfernen unwichtiger Dokumentteile. Der *Parser* schickt auf Grund des Inhaltes des PT Befehle zum *Document Object* die diesem mitteilen welche Elemente konstruiert werden müssen.

**Style** Bis der Stil eines Dokuments feststeht fließen Informationen aus sehr vielen Quellen ein. Die erste wichtige Quelle ist der Präsentationskontext, der die physikalischen Eigenschaften des Ausgabegeräts beschreibt (z. B. Drucker, Monitor). Daneben fließen die Standardeinstellungen des Browsers (*user-agent style sheet*), eventuelle *CSS*-Informationen[21] sowie explizite HTML-Stil-Eigenschaften ein (z. B. BOLD-*Tag*).

**Layout** Die *Layout*-Komponente bestimmt unter Zuhilfenahme der passenden Stil-Eigenschaften die Geometrie jedes Elements des Dokumrents. Das Resultat ist eine verschachtelte Hierarchie sogenannter *Flow Objects*[22]

**Rendering** Am Ende ist diese Komponente dafür zuständig die Umsetzung der Geometrie in einzelne Pixel durchzuführen, die auf dem Ausgabegerät dargestellt werden können.

**DOM** Das *W3C DOM Level* wird von Gecko komplett unterstützt. Die Umsetzung von *JavaScript*-Befehlen auf die *C++*-Schnittstelle, die Gecko bietet, erfolgt über einen *IDL-Compiler*[23]. Damit ist der komplette Dokument-Inhalt zugreifbar.

### 3.4 Mozilla ActiveX Control

Auch im Rahmen des Mozilla Projektes wird derzeit - analog zum *WebBrowser Control* von Microsoft (4.6) - ein *ActiveX Control* entwickelt, das die Funktionalität des *Gecko Layout Engine* kapseln und einfach verfügbar machen soll. Das Projekt befindet sich aber noch im Entwicklungsstadium.

Das *Control* bietet exakt die selbe Schnittstelle wie das *WebBrowser Control*. Die Entwickler von Mozilla haben dies so implementiert, um die Möglichkeit zu schaffen, beide Komponenten mit minimalen Änderungen am Code austauschen zu können.

Durch das *Mozilla Control* wird es auch möglich die umfangreichen und mächtigen API's, die Gecko mitbringt, in anderen Programmiersprachen, als C++[24] zu verwenden. Vorausgesetzt die Sprache unterstützt die Verwendung von *ActiveX*-Komponenten.

## 4 Microsoft Internet Explorer (IE)

### 4.1 ActiveX

Die *ActiveX*-Technologie wird deshalb in diesem Kapitel erörtert, da sie in der gesamten *Microsoft-Welt* eine äußerst wichtige Rolle spielt. Der folgende Abschnitt ist aber bestenfalls ein grober Über-

[21] Cascading Style Sheets
[22] kleinere geometrische Bereiche die zeilenweise wie ein *Stack* aufgebaut werden können.
[23] Interface Definition Language
[24] Die Gecko API's sind in C++ implementiert.

blick, um die Grundzüge von *ActiveX* kennenzulernen.

Eine *ActiveX* Komponente ist eine Datei, die ausführbaren Code enthält (z. B. eine EXE-, DLL- oder OCX-Datei). Der Code orientiert sich an der *ActiveX*-Spezifikation, die Art und Weise der Bereitstellung der gekapselten Objekte genau regelt. Die *ActiveX*-Technologie erlaubt es einem Entwickler, verschiedener dieser Komponenten zu sammeln, und sie in einer eigenen Applikation oder einem Dienst zu verwenden.

Zum einen gibt es *ActiveX*-Komponenten zu kaufen, die generische Dienste[25] zur Verfügung stellen. Die andere Möglichkeit ist die Entwicklung von Komponenten, die die eigenen Geschäftsprozesse unterstützen. Diese Komponenten können dann auch mit der *ActiveX*-Technologie mit kommerziellen Komponenten verknüpft werden. Die Verwendung von getestetem, standardisiertem Code in der oben beschriebenen Weise wird als *Component Software Development* (CSD) bezeichnet.

CSD mittels *ActiveX* darf man nicht mit objekt-orientierter Programmierung (OOP) verwechseln. OOP ist eine Methode um objekt-basierte Software-Komponenten zu erstellen; *ActiveX* hingegen ist eine Technologie, mit der die Verknüpfung verschiedenster objekt-orientierter Komponenten erreicht werden kann. Auf andere Weise ausgedrückt: OOP beschäftigt sich mit der Erstellung neuer Komponenten, während *ActiveX* das Ziel hat, eine Zusammenarbeit der Komponenten zu ermöglichen.

Nehmen wir mal an, ich hätte mit Hilfe von *Visual C++* eine Reihe nützlicher Objekte erstellt. Die Objekte könnten von jedem anderen C++-Entwickler verwendet und erweitert werden. Würde ich aber die Objekte in einer *ActiveX*-Komponente kapseln, so wäre die Verwendung und Erweiterung der Objekte nicht mehr alleine auf C++ beschränkt. Es wäre dann von jeder Programmierumgebung, die *ActiveX* unterstützt, aus möglich die Objekte zu nutzen und zu verändern.

## 4.2 Architektur des IE

Der IE benutzt *ActiveX Controls* und das *Active Document Interface* um seine Komponenten zu verbinden.

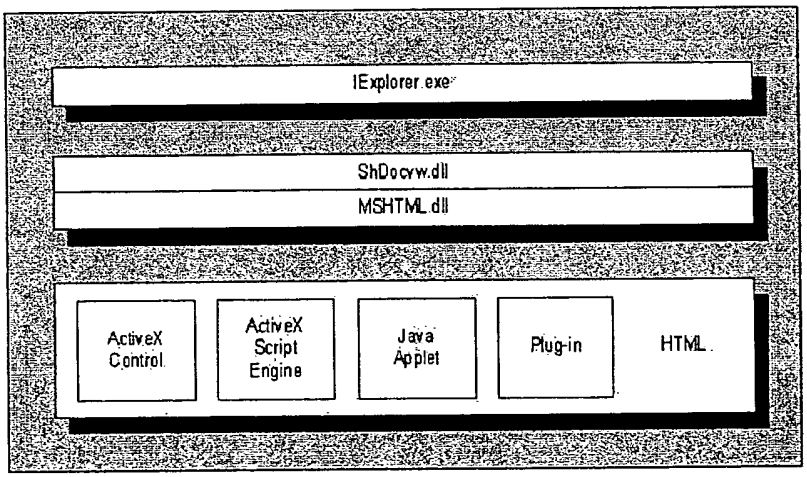



Abbildung 4: Die Architektur des Internet Explorer [3]

[25] z. B. Komponenten für numerische Analysen oder Elemente einer graphischen Benuzerschnittstelle

**IExplorer.exe** ist die Top-Level-Anwendung. Diese kleine Applikation wird instantiiert wenn der IE geladen wird. Sie verwendet die IE-Komponenten um folgende Aufgaben ausführen zu können: Navigation, *History*-Verwaltung, *Bookmark*-Verwaltung, das Parsen und Rendern von HTML-Code u. s. w. Diese Applikation ist auch für die *GUI*[26] des IE verantwortlich. Sie liefert u. a. die *Toolbars* und den *Frame*, in dem der IE dargestellt wird. *IExplorer.exe* beinhalted direkt die „Shdocvw.dll".

**Shdocvw.dll** beinhalted eine „Active Document"-Komponente. Im Normalfall ist dies die *Mshtml.dll*. Sie kann aber dynamisch durch eine andere solche Komponente (z. B. eine „Microsoft Office"-Applikation) ersetzt werden wenn der Benutzer ein Dokument mit entsprechendem Typ betrachten will. *Shdocvw.dll* bietet die Funktionalität, die für die Navigation, das Darstellen eines verlinkten Dokumentes im gleichen Browserfenster, *Bookmark*- und *History*-Management, sowie das Darstellen von Bildern benötigt wird.
Diese DLL bietet zudem eine Schnittstelle, die es erlaubt sie als *ActiveX Control* in andere Anwendungen einzubinden. Deshalb wird diese Komponente auch oft als *WebBrowser Control* bezeichnet.

**Mshtml.dll** ist die Komponente, die seit dem IE 4.0 für das Parsen und Rendern von HTML-Seiten zuständig ist. Daneben stellt sie auch noch das *DHTML Object Model* zur Verfügung. Innerhalb dieser DLL laufen die *Scripting Engines*, die *Microsoft Virtual Machine*, die *ActiveX Controls*, *Plug-ins* und andere Objekte, die von einer HTML-Seite referenziert werden könnten. Die Kommunikation mit der DLL geschieht über die COM[27]-Schnittstelle.

Die Architektur ist in einem Punkt rekursiv. Werden Frames verwendet, so gibt es in der obersten MSHTML-Instanz für jeden Frame eine SHDOCVW-Instanz, die alle wiederum eine MSHTML-Instanz enthalten.

## 4.3 ActiveX Scripting Engine

Durch das Konzept des *ActiveX Scripting Engine* (ASE) verwirklicht Microsoft seit dem IE Version 4.0 eine sehr offene und erweiterbare Umgebung für *Scripting Engines*. Es kann für buchstäblich jede Skriptsprache ein eigener SE gebastelt werden, der auch relativ einfach eingebunden werden kann. Außerdem können die Skriptsprachen in einem Web-Dokument relativ beliebig durcheinandergewürfelt werden, was ihre Interpretation nicht stört.

Um einen SE für eine noch nicht unterstützte Skriptsprache in das System zu integrieren muß lediglich eine entsprechende Parser-DLL installiert werden, die das *ActiveX Scripting Interface* implementiert. Über diese Schnittstelle kann der ASE auf die DLL zugreifen. Natürlich treten Probleme auf, wenn die Skriptsprache auf einem anderen Browser interpretiert werden soll. Es muß also in diesem Fall sichergestellt werden, daß der SE auch auf anderen Browsern installiert ist, oder installiert werden kann.

## 4.4 Browser Extensions

Die Microsoft *Browser Extensions* ermöglichen es Entwicklern auf einfache Art und Weise Zugriff auf ihre eigenen Browser-Erweiterungen zu gestatten, indem sie die Standard-Benutzerschnittstelle mit eigenen Elementen (z. B. ein *Explorer Bar*) erweitern können. Dieses Feature erlaubt es seit dem IE version 4.0 *Explorer Bars* zu integrieren, und das Kontext-Menü[28] mit eigenen Einträgen zu ergänzen. Seit dem IE Version 5 ist es zusätzlich möglich Einträge in das *Tools*-Menü einzufügen, und es können eigene *Buttons* in die *Toolbars* eingefügt werden. Seit der Version 5 gestatten die *Browser*

[26] Graphical User Interface
[27] Component Object Model
[28] Normalerweise erreichbar über die rechte Maustaste.

*Extensions* auch den Zugriff auf Win32-Applikationen und externe Skripte. Dieser Zugriff kann über Menüeinträge und zusätzliche *Buttons* zur Verfügung gestellt werden.

Um beipielsweise einen *Button* auf der Oberfläche einzufügen braucht man Kenntnisse über die *Windows Registry* und *Globally Unique Identifiers* (GUID). Der *Button* wird über eine ganze Latte von *Registry*-Einträgen festgelegt und beschrieben. Programmierkenntnisse sind eigentlich nicht erforderlich.

### 4.5 Shell Helper API

Seit dem IE Version 5 bietet Microsoft die *Shell Helper API*, die u. a. für die Bookmark-Verwaltung zuständig ist. Das Neue daran ist die Möglichkeit *Bookmarks* über das Internet auszutauschen und zu synchronisieren.

Jeder kennt das Problem, daß sich ein entsprechender *Bookmark*-Eintrag immer genau auf dem Rechner befindet, auf dem man gerade nicht arbeitet. Mit dem Konzept, das die *Shell Helper API* bietet wird es z. B. für ISP's[29] möglich eine Ablage der *Bookmarks* im Internet anzubieten, wo sie von jedem Rechner aus einfach zugreifbar wären.

Sobald der Benutzer dann im IE seine Favoriten-Liste aufruft wird die Liste automatisch mit der Liste im Internet synchronisiert, wodurch er immer die aktuellen *Bookmarks* auf dem aktuellen Arbeitsrechner zur Verfügung hätte.

### 4.6 WebBrowser Control

Wie schon oben erwähnt kann viel der Funktionalität, die der Internet Explorer mitbringt, über ein *ActiveX Control* in eigene Applikationen eingebunden werden. Einer Applikation, die das *Control* verwendet, steht fast die komplette Funktionalität des IE zur Verfügung: Ausführung von Skripten, DOM, DHTML, Java, das Darstellen von HTML-Seiten, XML u. s. w.. Es wird die selbe Oberfläche gezeigt, die auch im IE sichtbar ist. Lediglich das umgebende Fenster fehlt.

Die Komponente kann dann objektorientiert über Methoden und Attribute gesteuert und abgefragt werden. Zudem bietet die Komponente auch noch ein *Event Model*, das z. B. vor dem Darstellen einer neuen Seite ein Ereignis auslöst, auf das dann entsprechend reagiert werden kann.
Das *WebBrowser Control* ist auf jedem Windows-System verfügbar, auf dem ein IE installiert ist.

## 5 Zusammenfassung und Ausblick

Die Nutzung von Web-Browsern an der Schnittstelle zwischen Informationssystemen und Benutzer bietet viele Vorteile gegenüber der Verwendung spezialisierter Anwendungen. Mit den heute verfügbaren Browser-Technologien kann - wenn bestimmte Rahmenbedingungen erfüllt sind[30] - die komplette Funktionalität einer „normalen", auf einem Rechner laufenden, proprietären Anwendung nachgebildet werden.

Ein Vorteil der Webtechnologie ist, daß als Basisausstattung nur ein Browser benötigt wird, um auf ein Informationssystem zugreifen zu können. Diese Voraussetzung ist heute wahrscheinlich schon auf den meisten Rechnern erfüllt. Zu den einfachen Mindestbedingungen kommt der gewohnte Umgang mit dem Browser, was heute auch als gegeben angenommen werden kann.
Diese einfache Infrastruktur bedeutet eine konkrete Kostenersparnis, z. B. für ein Unternehmen.

[29] Internet Service Provider
[30] Kontrolle über die Browser-Konfiguration des Benutzers, z. B. im *Intranet*

Ein weiterer Vorteil, der Nutzung von Web-Browsern als Zugangssoftware, ist die Skalierbarkeit solcher Systeme. Ein Informationssystem, das bereits im *Intranet* verfügbar ist kann im Normalfall mit geringem Aufwand auch im *Internet* verfügbar gemacht werden.

Es gibt allerdings einen Unterschied, in welchem Grad die Möglichkeiten, die Web-Browser heute bieten, eingesetzt werden können.
In einem System das lediglich im *Intranet* eingesetzt wird gibt es so gut wie keine Einschränkungen. Dies liegt daran, daß die relevante, eingesetzte Software komplett der System-Administration bekannt ist und kontrolliert werden kann. In solch einem Umfeld können auch neue Technolgie-Features schnell mit Erfolg eingesetzt werden.
Im Internet bietet sich eine andere Situation. Wird ein Informationssystem im Intenet verfügbar gemacht, so muß genau abgewägt werden welche Technolgien bei den meisten Benutzern erwartet werden können. Es wäre wenig vorteilhaft ein brandneues Portal ins Internet zu stellen, das aber nur 5Internet-Besucher nutzen können, da der Rest nicht über ausreichende ausgestattete Browser verfügt. Dies sind einige der kritischen Technologien: Cookies, JavaScript, Java, Verwendung seltener Scriptsprachen, CSS, die Annahme der Benutzer habe bestimmte Plug-ins installiert, usw.

Auf Grund der genannten Vorteile - vor allem der Einfachheit und Verfügbarkeit - ist es aber zu erwarten, daß die Verwendung von Web-Browser als Zugangsmedium immer wichtiger werden wird. Der aktuell sehr stark wachsende Bereich der *Application Service Provider* (ASP)[31] ist ein Indiz hierfür. Für die Zukunft kann auch mit Systemen gerechnet werden, die noch einen Schritt weiter gehen und nur noch einen Web-Browser installiert haben. Das System besitzt dann über keinen Plattenspeicher mehr, und alle Anwendungen sind auschließlich *online* verfügbar.

[31] Anbieter, die Applikationen über das Internet zur Verfügung stellen. Die Programme sind nicht lokal installiert, sondern werden über den Web-Browser genutzt und gesteuert.

## Literatur


[1] Rick Gessner. Netscape's Gecko - The Next-Generation Layout Engine. Technical report, Netscape, 1999.
http://www.webtechniques.com/archives/1999/03/gessner/.

[2] Microsoft MSDN Library. DOM Overview.
http://msdn.microsoft.com/workshop/Author/dom/domoverview.asp.

[3] Microsoft MSDN Library. Web Workshop - Browser Overview.
http://msdn.microsoft.com.

[4] Mozilla-Projekt. Homepage.
http://www.mozilla.org.

[5] Netscape. Developer Homepage.
http://developer.netscape.com.

[6] World Wide Web Consortium (W3C). Homepage.
http://www.w3c.org.